Design of Synthesis-Ready Core of Fixed-Point Arithmetic Block

A Thesis Submitted

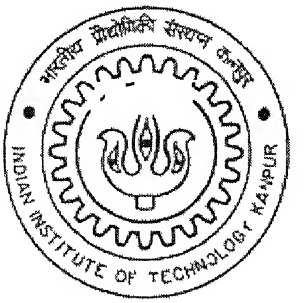
In Partial Fulfillment of the Requirements

For the Degree of

Master of Technology

By

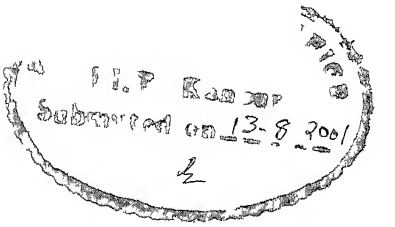
Reetesh Chaurasia



Nuclear Engineering and Technology Programme

Indian Institute Of Technology Kanpur

August 2001



Certificate

It is certified that the work contained in the thesis entitled **Design of Synthesis-Ready Core of Fixed-Point Arithmetic Block**, by Reetesh Chaurasia has been carried out under my supervision and that this work has not been submitted elsewhere for a degree

Dr B Mazhari
Electrical Engineering Department
Indian Institute of Technology Kanpur
August 2001

Abstract

Advent of standard procedures and competent CAD tools has facilitated hardware designing cope with the demand for digital systems with growing complexity. Traditional schematic-based design methodologies have given way to Hardware Description Language (HDL). Very High Speed Integrated Circuits HDL (VHDL) can be used for documentation, verification, and synthesis of large digital systems and supports different levels of abstraction for describing the hardware. Programmable devices like Field-Programmable Gate Array (FPGA) and VHDL can be used in tandem, to explore large design space in less time.

The present work comprises the development of core design of fixed-point arithmetic block. The model supports basic arithmetic operations of addition, subtraction, multiplication, and division of fixed-point binary numbers in Q15 format and is extendable to Q31 format arithmetic operations. The work involves modeling and simulating different algorithms for arithmetic operations with VHDL and verifying them using test bench. The work essentially encompasses a comparative study of FPGA implementations of various techniques for binary arithmetic operations. The selection of a technique, to be incorporated in the final design, is based on the results of the implementation. The generality and portability is maintained by avoiding any vendor-specific optimized block in the design.

Acknowledgement

I esteem it my privilege to express my deep sense of gratitude to my thesis supervisor Dr B Mazhari, for his valuable guidance and criticisms, which provided necessary stimulus for the successful completion of the work

I am grateful to Dr P Munshi for always being supportive to me His help, academically and otherwise, is gratefully acknowledged

I thank my friends and classmates who made my stay at the institute a memorable one

Reetesh Chaurasia

Contents

List of Figures

Chapter 1

Introduction

This section presents an overview of the work It introduces the basic motivation of the work along with the state of the art The brief objective of the work and approach towards it are also outlined in the chapter The chapter is divided in the following sections

- Overview
- Objective
- Approach
- Outline

Overview

The growing sophistication of digital systems invariably requires the presence of both hardware and software components in the system The complexity vs performance considerations of the design, dictates the choice of implementation i e , hardware or software, for a particular operation Software is easy to create, debug, and modify but are slow and difficult to maintain as compared to the hardware This very contrast in features of software and hardware is exploited to

Chapter 1

Introduction

This section presents an overview of the work It introduces the basic motivation of the work along with the state of the art The brief objective of the work and approach towards it are also outlined in the chapter The chapter is divided in the following sections

- Overview
- Objective
- Approach
- Outline

Overview

The growing sophistication of digital systems invariably requires the presence of both hardware and software components in the system The complexity vs performance considerations of the design, dictates the choice of implementation i e , hardware or software, for a particular operation Software is easy to create, debug, and modify but are slow and difficult to maintain as compared to the hardware This very contrast in features of software and hardware is exploited to

the fullest in designing the hardware itself Short turnaround time has become critical and competent CAD tools have evolved for the purpose Software programmable components are used extensively as they facilitate quick design revisions yet maintaining the merits of hardware

Most of the digital systems are built around a central processing unit (CPU), the function of which is to execute a sequence of instructions The sequence of operations involved primarily consists of two phases the fetch cycle and the execution cycle The instruction is obtained from the main memory in the fetch cycle and the execution cycle includes decoding the instructions, fetching any required operands, and performing the specified operation At the outset, a CPU consists of two units Program control unit and the data processing unit The program control unit handles the fetch cycle and a part of execution cycle, while the data processing unit operates over the operands The core of the data processing unit is the arithmetic-logic unit (ALU), which performs the desired arithmetic or logic operations

The performance of the processor depends on the constitution of the arithmetic-logic unit The capabilities of the ALU largely outline the performance indices of the processor Many microprocessors can perform only fixed-point addition and subtraction e g , AMD 2901 16-bit ALU and which severely limit the arithmetic performance of the processor The present work is an approach to design the arithmetic unit, which can perform all the four basic arithmetic operations The algorithms implemented in the design for the arithmetic operations, are the governing factors for the competence of processor Different algorithms have different implications in term of area and speed of operations, and optimal choice can only be made after a comparative study The present work involves a comparative study of various algorithms for fixed-point binary arithmetic

the fullest in designing the hardware itself Short turnaround time has become critical and competent CAD tools have evolved for the purpose Software programmable components are used extensively as they facilitate quick design revisions yet maintaining the merits of hardware

Most of the digital systems are built around a central processing unit (CPU), the function of which is to execute a sequence of instructions The sequence of operations involved primarily consists of two phases the fetch cycle and the execution cycle The instruction is obtained from the main memory in the fetch cycle and the execution cycle includes decoding the instructions, fetching any required operands, and performing the specified operation At the outset, a CPU consists of two units Program control unit and the data processing unit The program control unit handles the fetch cycle and a part of execution cycle, while the data processing unit operates over the operands The core of the data processing unit is the arithmetic-logic unit (ALU), which performs the desired arithmetic or logic operations

The performance of the processor depends on the constitution of the arithmetic-logic unit The capabilities of the ALU largely outline the performance indices of the processor Many microprocessors can perform only fixed-point addition and subtraction e g , AMD 2901 16-bit ALU and which severely limit the arithmetic performance of the processor The present work is an approach to design the arithmetic unit, which can perform all the four basic arithmetic operations The algorithms implemented in the design for the arithmetic operations, are the governing factors for the competence of processor Different algorithms have different implications in term of area and speed of operations, and optimal choice can only be made after a comparative study The present work involves a comparative study of various algorithms for fixed-point binary arithmetic

Synthesis of the verified designs over field-programmable gate array (FPGA) yields the area and timing results

Objective

The objective of the work is to develop a synthesis-ready, tested core design of a fixed-point arithmetic unit in Q15 format Objective further lays the requirement that the design has to be configurable to be used with Q31 format numbers The arithmetic block is required to incorporate the support for all four basic arithmetic operations addition, subtraction, multiplication, and division Per the requirements, the behavioral modeling, the RTL designing and the test-bench verification is in VHDL The test-bench accepts the test vectors from file on disk Further, in order to keep the design portable, it must not contain any vendor-specific pre-optimized block Synthesis and implementation are to be done with FPGA, which also decides the limit on the operating speed of the unit

Approach

A top-down design methodology is adhered to in the present work At the outset, a behavioral model of the arithmetic unit is first derived The tool used for the purpose is Active HDL editor 4 0 by Aldec The design environment complies with IEEE1076 VDHL'93 standards Test-bench verification, for the correctness of behavioral model, is done using the same tool The performance of the arithmetic unit depends strongly on the algorithm used for the arithmetic operations and so a comparative study of implementations of various binary

arithmetic operations is performed A particular algorithm for fixed-point binary arithmetic operation is first entered in VHDL and is verified using test-bench For synthesis, a netlist is extracted using the design environment Standard electronic design interface format (EDIF) netlist is used in the present work FPGA synthesis yields the timing and equivalent gate count results, which are then used to establish a comparison between the different implementations Finally, an optimal choice for the implementation of arithmetic operation is made based on the results of the study

Outline

Chapter 1 presents an overview of the work It introduces the basic motivation of the work along with the state of the art The brief objective of the work and approach towards it are also outlined in the chapter

Chapter 2 presents the theory for digital system designing Starting from the fundamentals of digital arithmetic operations and other relevant aspects, it deals with the today's design methodology It introduces the HDL design entry and the FPGA synthesis It includes the details of various algorithms for fixed-point binary arithmetic

Chapter 3 contains the methodology followed towards the designing of the arithmetic unit It presents the strategy in the most detailed perspective It includes the HDL modeling and FPGA synthesis of various implementations

apter 4 presents the comparative results of implementations of various binary hmetic operations It also includes discussion to make an optimal choice vards the objective

apter 5 contains the conclusion and outlines the scope of further work

Chapter 2

Theory

This section introduces the basic theoretical aspects of the digital system designing and elaborates upon the fundamentals utilized in the work It encompasses arithmetic aspects of fixed-point binary numbers in addition to the VHDL modeling test-bench verification and FPGA synthesis fundamentals The chapter has the following sections

- Design process
- Q15 format arithmetic
- VHDL modeling
- FPGA synthesis
- Binary arithmetic implementation

Design Process

Most of the logic circuits, based on gates and flip-flops, have been traditionally designed with Boolean equations Many techniques were developed to optimize this methodology, like minimization of equations for efficient usage of components Schematic-based design methods expanded the capabilities of

Boolean equations by accepting hierarchical design entries and thus accommodating larger number of components in the design But with the ever-increasing demand for high densities, the schematic-based technique became insufficient New technologies evolved to address the demand In the last four years HDL based being used for less than 5% of new designs grew up to nearly 100% of all designs CAD tools developed for HDL modeling provide and integrated platform addressing a spectrum of designing tasks Modern synthesis tools are developed in a manner that provides a seamless translation from design specifics to place-and-route

The design of digital systems, especially VLSI systems can be divided into the following steps

- Design specification The desired behavior of the system is specified at some level of abstraction It comprises of a set of requirements and is silent about the design
- High-level design This level transforms the design specifications into a description that uses library components like memories, controllers etc This description is more precisely caller register-transfer level (RTL) description
- Logic design The RTL description is first optimized for an objective function defined by performance constraints This steps is also called the logic optimization and yields the final implementation in terms of primitive cells in the library like gates, registers etc
- Physical design At this step, locations of various modules are determined and interconnections between them are routed, owing to which the step is also called as place-and-route The final layout is ready for fabrication

Some of these steps are to be iterated in order to obtain an acceptable level of match between the desired and actual performance indices With the growing

complexity of the digital circuits, it has becomes indispensable to use CAD tools In addition to the speed, they also explore a larger design space and yield potentially better designs In the present work different CAD tools are used for the respective purposes

The new design methodology can be represented by the following design cycle

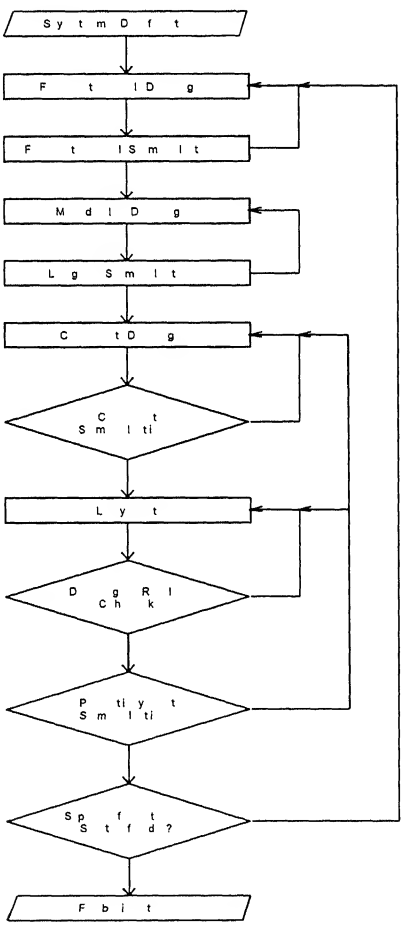


Figure 1 The Design Flow

The above design cycle depicts the various steps in the process of designing along with the respective tools required for the purpose At every identifiable step in the process, a simulation or a relevant check is performed to catch the error The longer the error stays undetected, the more costly it will be to correct

Q15 Format Arithmetic

Digital computation has finite precision, resulting in the approximation of parameters Finite-length arithmetic is any arithmetic with fixed or limited precision Both floating-point and fixed-point formats are considered finite-length Floating-point software and hardware set the numerical size without much intervention from a programmer To work with the finite precision power of the fixed-point formats, each numerical operation has to be verified to avoid overflow and truncation of the fractional portion of the number Generally, DSP families have a 16-bit data word, and this word resolution is great enough for most real-world problems

A fixed-point number consists of a word-length, which is the total number of bits in the representation, divided into integer and fractional parts This division of word-length decides the precision and the range of binary representation

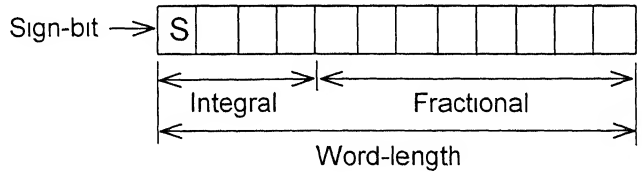


Figure 2 Fixed-Point Number Format

Fixed-point arithmetic is defined by Q formats For example, the Q15 (1 15) format is a popular format in which the most significant bit is the sign bit followed by an imaginary binary point, followed by 15 bits of fraction or mantissa normalized to 1 The preset work deals with the arithmetic operations in Q15 format

The precision of the fixed-point binary number in a particular format, which is the smallest difference between two numbers in the same format, is governed by the number of digits in the fractional part of the number For Q15 number it is 2^{15}

The range of the fixed-point binary number depends whether the representation is signed or unsigned The unsigned Q15 number has a range $[0, 2 - 2^{15}]$ while the signed representation has a range $[-1, 1 - 2^{15}]$

A fixed-point binary number in signed Q15 format can be expressed as

$$h = \sum_{n=0}^{15} b_n \, 2^{-n} \quad or,$$
$$h = -b_0 \, 2^0 + b_1 \, 2^{-1} + b_2 \, 2^{-2} + \quad + b_{15} \, 2^{-15}$$

Where, h is the fractional number and b_i is the i th bit value in the representation

Addition and subtraction using Q15 numbers is simple using digital hardware Multiplication in digital hardware, however, requires special understanding When two 16-bit numbers are multiplied, the product is a 32-bit number If the input formats of the numbers are X, Y and W, Z, the product has the format (X+Y) (W+Z)

Product rule		*Binary example*
Operand A	X Y	1 111 [1 3 format]
Operand B	× W Z	× 1 111 [1 3 format]
Result	(X+W) (Y+Z)	11 000001 [2 6 format]

A product of two Q15 (1 15) numbers would give a Q30 (2 30) number with 2 sign bits and 30 bits of fraction Because the second sign bit is redundant, the Q30 result should remain shifted by one bit The result will be Q31 (1 31), and the final result will be in the upper 15 bits of the product In the present work, this method of truncation is utilized for the multiplication of two Q15 operands

VHDL Modeling

VHDL is an acronym, which stands for VHSIC Hardware Description Language VHSIC is another acronym, which stands for Very High Speed Integrated Circuits VHDL is a standard (VHDL-1076) developed by the IEEE The language has been through a couple of revisions, the most widely used version is the 1987 (Std 1076-1987) version, sometimes referred to as VHDL'87 or just VHDL However, there is a newer revision of the language referred to as VHDL'93 VHDL'93 is fairly new and is in the process of replacing VHDL'87 The design environment used in the present work is VHDL '93 compliant VHDL can be used for documentation, verification, and synthesis of large digital designs This is actually one of the key features of VHDL, since the same VHDL code can achieve all three of these goals, thus saving a lot of effort and reducing the introduction of errors between translating a specification into an implementation

In addition to being used for each of these purposes, VHDL can be used to take three different approaches to describing hardware These three different approaches are the structural, data flow, and behavioral methods of hardware description Most of the time, a mixture of the three methods is employed and a complete design will have different sections expressed in different ways The

terms algorithmic and RTL (Register Transfer Language) are sometimes used for behavioral and dataflow

Structural description

The structural description of a design is simply a textual description of a schematic In VHDL every portion of a VHDL design is considered a block A VHDL design may be completely described in a single block, or it may be decomposed in several blocks Each block in VHDL is analogous to an off-the-shelf part and is called an entity The entity describes the interface to that block and a separate part associated with the entity describes how that block operates The interface description is like a pin description in a data book, specifying the inputs and outputs to the block The description of the operation of the part is like a schematic for the block

A block is a design, and a complete design may be a collection of many blocks interconnected This is achieved by defining the blocks as components and then instantiating them as and when required

Data flow description

In the data flow approach, circuits are described by indicating how the inputs and outputs of built-in primitive components or pure combinational blocks are connected together In other words, it describes how signals (data) flow through the circuit The entity-architecture pair describes a functional block in the data flow description The entity describes the interface to the design while the architecture part describes the internal operation of the design

The data flow describes how the data flows from inputs to outputs In VHDL this is accomplished with the signal assignment statement VHDL supports signal assignment statements through expressions

Behavioral description

The behavioral approach to modeling hardware components is different from the other two methods in that it does not necessarily in any way reflect how the design is implemented It is basically the black box approach to modeling It accurately models what happens on the inputs and outputs of the black box, but what is inside the box is irrelevant

The behavioral description is usually used in two ways in VHDL First, it can be used to model complex components that would be tedious to model using the other methods Second, the behavioral capabilities of VHDL can be more powerful and is more convenient for some designs In this case the behavioral description will likely imply some structure of the implementation Also in the early part of the design process behavioral models are used to quickly evaluate an algorithm and simulate it by passing real data through it
In the present work, a top-down design approach is followed where each block in the system is described with a working behavioral model until it can be implemented structurally

Lastly, there is a trend to use synthesis tools to automate the conversion of behavioral to structural, with restraints on the target technology and this further emphasizes the effectiveness of behavioral modeling

Simulation

The purpose of simulation is to gather the information about changes in the system state over time Once the structure and behavior of a module have been specified, it is possible to simulate the module by executing its behavioral description It allows a design to be simulated before being manufactured This facilitates a quick comparison of design alternatives and test for correctness without the delay and expenses for hardware prototyping

The scheme used to simulate a VHDL design is called discrete event time simulation When the value of a signal changes, an event is said to have occurred on that signal This is the foundation of the discrete event time simulation The values of signals are only updated when certain events occur and events occur at discrete instances of time One event result in another, so simulation proceeds in rounds The simulator section of the design environment maintains a list of events that need to be processed In each round, all events in a list are processed, any new events that are produced are placed in a separate list and are scheduled for processing in a later round

The simulation starts with an initialization phase where all the signals assume their initial value and simulation time is set to zero All the behavioral modules are then executed Afterwards, simulation proceeds in two stages In the first stage, the simulation time is advanced to the earliest time at which the transaction has been scheduled All transactions scheduled for that time are executed, which in turn may cause events to occur on some other signals In the second stage, all modules that react to events occurring in the first stage have their behavior executed When all of the behavior programs have finished executing, the simulation cycle repeats

Test bench

A test bench is an environment where a design is checked for correctness by applying signals and monitoring the responses by observing the signal probes and monitors A test bench substitutes the design environment in such a way that the behavior of design can be observed and analyzed

In VHDL, test bench is an integral part of the design environment and is a specification simulated by the simulator It consists of the unit under test (UUT), and processes supporting the stimuli applied to the UUT Stimuli for UUT can be specified inside the test bench or can be in the file on the disk The response of the UUT, on the other hand, can be observed through the simulator outputs

The internal states of the system, specified by the signals used in modeling, can be monitored in a number of ways depending on the CAD tool In the present work, a waveform editor serves the purpose All the signals can be monitored in the form of a waveform over the simulation time This feature facilitates the use of test bench and makes it more interactive

FPGA Synthesis

Short turnaround time has become critical in the design of digital systems Software programmable components like microprocessors and digital signal processors have been used extensively in the systems since they provide quick design revisions However, the inherent performance limitations of the software-programmable systems make them inadequate for high performance designs As a result mask-programmable gate arrays (MPGA) evolved MPGA consist of rows of transistors that can be interconnected to implement the desired circuit

The mask layer that defines the logic of the chip is prefabricated and metal layer are customized to connect the transistors to appropriately implement the desired circuit Although MPGA provided significant performance enhancement, they curtailed flexibility of user programmability Moreover, manufacturing time was a bottleneck

The user-programmable hardware devices are prefabricated arrays of identical programmable logic blocks with routing resources and can be configured to the desired circuit with configuration time of the order of minutes Also being off-the-shelf component they provide a cheaper alternative to MPGA User-programmable hardware devices can be broadly classified into two categories namely the programmable logic devices (PLD) and field-programmable gate arrays (FPGA)

PLD typically consists of interconnections of programmable logic arrays (PLA) A PLA has two planes – an AND plane and an OR plane The AND plane implements the product terms and the OR plane realizes their summations FPGA are like MPGA that they have uncommitted logic blocks and interconnect However, they offer the flexibility of user programmability like PLD FPGA differ from PLD in the logic block granularity, FPGA have more fine-grain logic block A logic block is a versatile configuration of logic elements that can be programmed by the user

There are two popular categories of FPGA block structures, namely look-up table based (LUT) and multiplexor based, the resulting architectures are called LUT-based and MUX-based architectures respectively In the present work Xilinx 4000 series device is used for synthesis, which is a LUT-based FPGA The basic block of LUT architecture is a look-up table that can implement any Boolean function of up to m inputs, $m \geq 2$ For a given LUT m is a fixed number typically

between 3 and 6 A *m*-LUT is typically implemented by static random access memory (SRAM) that has m address lines and 1 data line

Xilinx FPGA

The basic building blocks for Xilinx FPGA is shown below

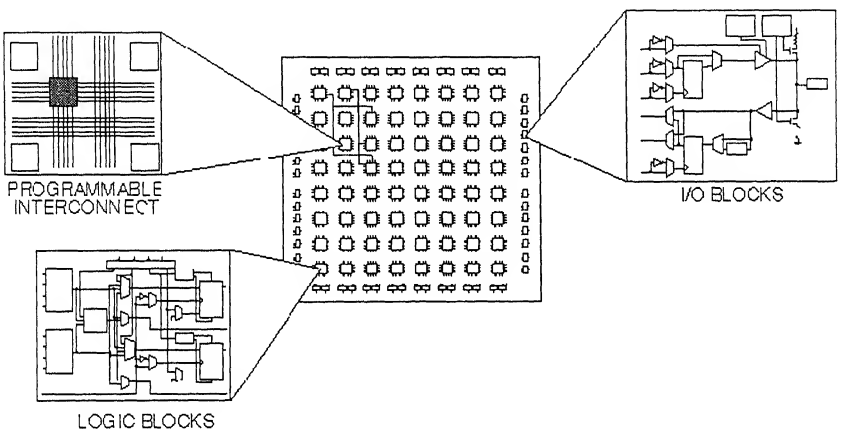


Figure 3 Constitution of Xilinx FPGA

Configurable Logic Block

The basic cell, called a combinatorial logic block (CLB), is made up of three function generators F, G, and H The F and G generators are basically just RAM blocks with four address bits and one data bit hence, a 16x1 RAM, which stores the combinatorial behavior of the given inputs The H function generator has three inputs – output of F, output of G, and a third input from outside the logic block The CLB can implement functions of up to nine variables with this arrangement In addition two flip-flops are available which can store the function generators outputs The architecture allows the flip-flops to be used independently and has Set-Reset circuitry The outputs are Y, YQ, X and XQ As can be seen in the figure below, X can output F or H outputs and Y can output G or H outputs

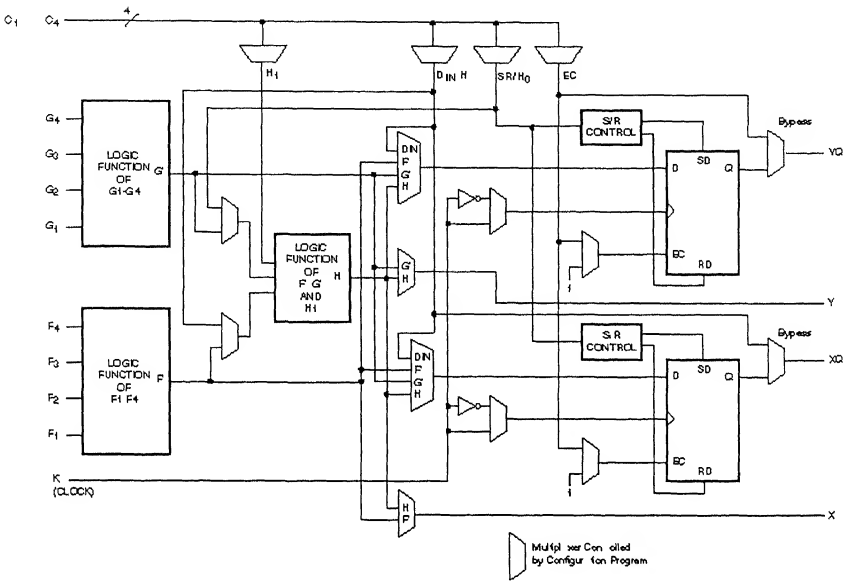


Figure 4 Xilinx CLB

Input Output Block

The input-output blocks (IOB) signify are how the FPGA interfaces to its pins or pads With reference to the figure below, I1 and I2 bring in signals from outside the FPGA As can be seen each IOB has an input and output, flip-flop and an input and output buffer Other features of the IOB are Inputs can be delayed by several nanoseconds to compensate for clock delays and the outputs can be inverted or not Buffers can be bypassed to eliminate delay

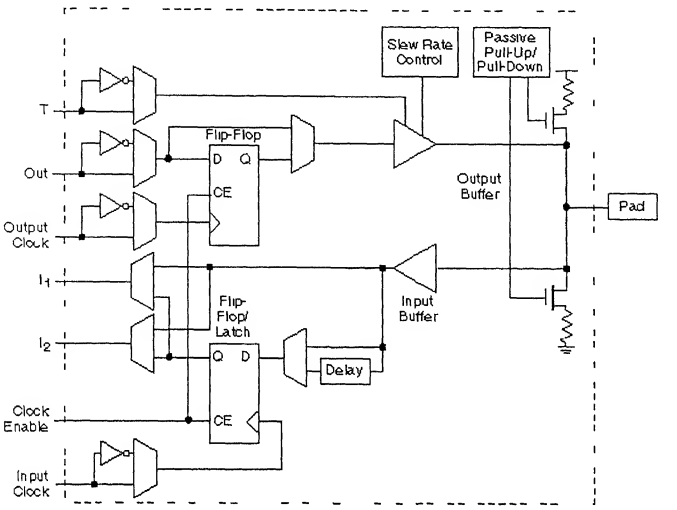


Figure 5 Xilinx input-output block

Programmable Interconnects

All internal connections are made up of metal segments with programmable switching points to implement the routing There are three main interconnect types as stated below

Single length lines a grid of horizontal and vertical lines, which intersect at a switch matrix between each CLB

Double length lines a grid of horizontal and vertical lines, which intersect at the switch matrixes between two CLBs

Long lines forming a grid of metal interconnection segments that runs through the entire length or width of the array Global buffers can drive additional vertical lines designed to distribute clock signals and other high fan-out signals

The router software uses all these elements, to take the synthesized design and place it in the desired FPGA part Understanding the architecture of an FPGA is

very important when optimizing designs In the present work, all the designs were optimized using the Xilinx 4000 series FPGA

Binary Arithmetic Implementations

The present work demands the arithmetic unit to support four basic arithmetic operations of addition, subtraction, multiplication, and division over the Q15 format operands There could be several possible alternatives to implement a module performing a particular arithmetic operation Out of these, the one with optimal area and delay has to be selected for each of the four operations

Every implementation of the arithmetic operation has it's own merit and demerits, which would show up after synthesizing the design Nevertheless, certain comparative estimates can me made beforehand as to what the final outcome will be In the work, the implementation considered for arithmetic operations are enumerated as under

- Addition/Subtraction
    - Ripple carry adder
    - Carry look-ahead adder
    - Carry-skip adder
- Multiplication
    - Shift-and-add multiplier
    - Carry-save multiplier
    - Booth-Wallace multiplier
- Division
    - Array divider

Q15 Binary Addition/Subtraction

Binary addition and subtraction can be performed using the same hardware In the present work, Q15 numbers in 2's complement representation are considered A brief summary of implementations considered in the work is given below

Ripple Carry Adder

A ripple carry adder is an adder where the carry bit ripple through all the bits from right to left It is similar to normal decimal addition It comprises of a serial arrangement of *n* full-adders, where *n* is the data width The schematics of a ripple carry adder are shown below

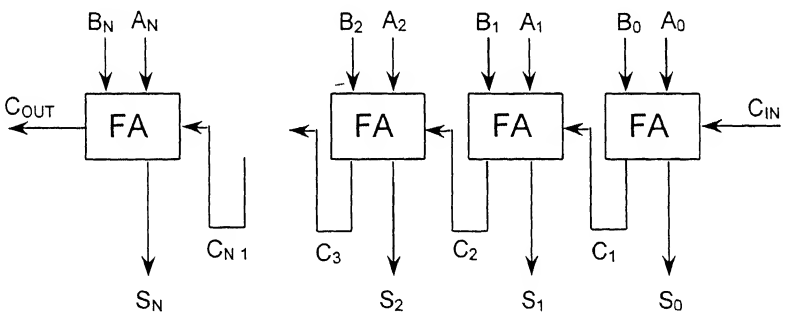


Figure 6 Ripple Carry Adder

The implementation is done through component instantiation of full adder units It is the simplest, smallest and the slowest of all carry propagate adder implementations

Carry Look Ahead Adder

The long time for the carry to ripple through all the bits in a ripple carry adder has to be minimized One way, to achieve this is to propagate the carry to the left as

fast as possible A carry look ahead adder splits the carry into two parts namely the propagate carry and the generate carry For each stage of full adder these two bits accompany the sum bit Four such modified full adder units together give input to a carry look-ahead unit For Q15 addition/subtraction, four such carry look-ahead units feed one other unit of same type The schematics of a 16-bit carry look-ahead adder is shown below

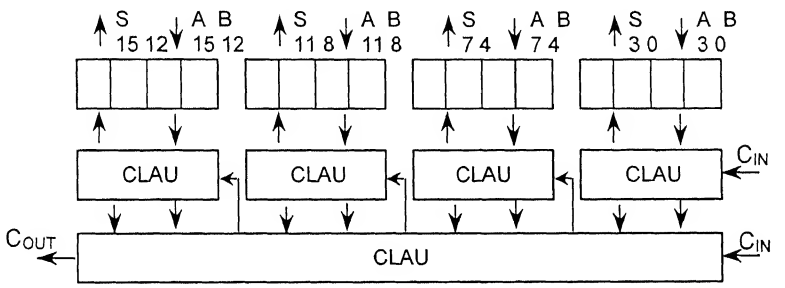


Figure 7 Carry Look Ahead Adder

Carry look-ahead adder provides a speed-up for a medium overhead in hardware

Carry-Skip Adder

Carry-skip addition is a modification over the ripple carry adders in terms of medium speed-up at the cost of a medium hardware overhead It basically concatenates the partial ripple carry blocks and generates the carry for such a block based on carries generated in the previous stages The schematic for a carry-skip adder is shown below

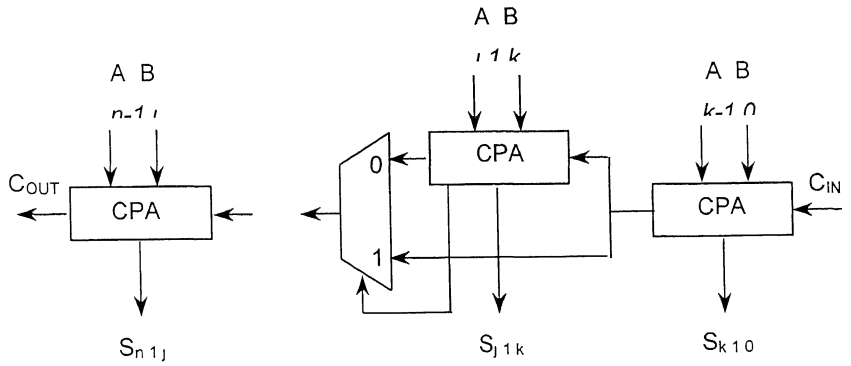


Figure 8 Carry-Skip Adder

This implementation has inherent logic redundancy, which poses problems in design optimization, timing analysis, and testing

Q15 Binary Multiplication

The multiplication algorithms for binary numbers are similar to the normal decimal multiplication Partial products are computed and are added to the ones calculated at the previous levels with appropriate shifts given to them Speed-up is possible but that requires a certain overhead in hardware A brief note on the implementations considered in the present work is given below

Shift-and-Add Multiplication

It is the simplest form of multiplication, much like the normal paper-pen multiplication approach For Q15 multiplication, 16 partial products are computed and added with shift It is not required to hold all the partial products till the end and then add all of them in one go Rather, in order to save hardware, at a time the partial product computed is given a shift and added to the stored sum

Likewise, in 16 shift and add steps the process completes The schematics of the shift-and-add multiplication is shown below

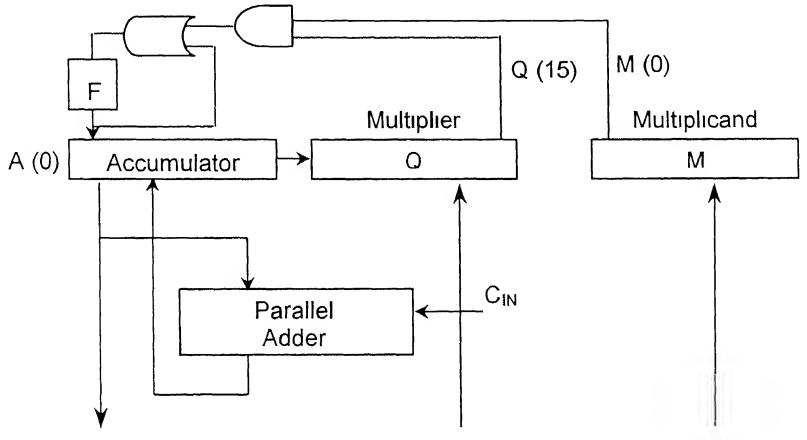


Figure 9 Shift-and-Add Multiplier Block

Though, the constitution of the shift-and-add multiplier is fairly simple and easy to optimize, to multiply two n-bit numbers, n clock cycles are required This makes the shift-and-multiplier slower than other high-speed multiplier implementations

Carry-Save Multiplication

It makes use of carry save addition to compute the product The concept of carry save addition is to save the carry for the next step Partial products are computed using the array of full adder blocks This multiplier is also called the array multiplier The schematics of the multiplier are shown below

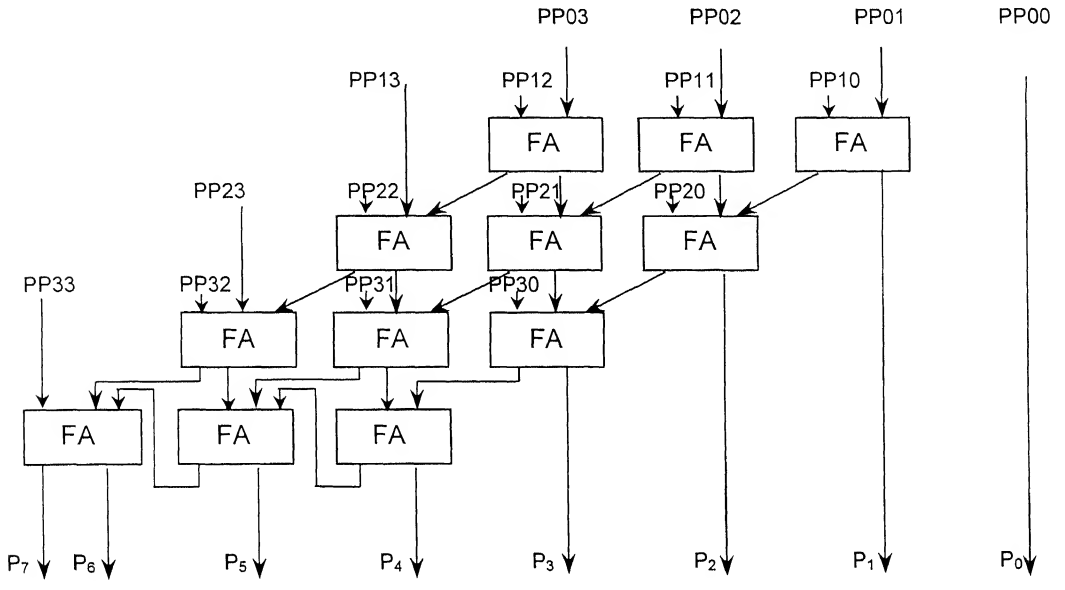


Figure 10 Carry Save Multiplier

To multiply two n-bit numbers, a carry save multiplier requires $n*(n-1)$ full adders The longest delay will be the one coming from the first partial product, going down the middle adder rows and then rippling though all the full adders of the last row This makes the implementation costly in terms of hardware and delay

Booth-Wallace Multiplication

This multiplication implementation is based on the Booth recoding scheme, proposed by Andrew Booth and all the additions involved in the process are done through the tree adders proposed by Wallace The idea is to recode a group of bits from the multiplicand and then determine the partial product The partial products obtained after recoding can be ± one or two times the multiplicand The corresponding weighted values of the partial products are then added using the Wallace tree adders The recoding scheme used in the present work is given below

Recoding Bits	Partial Product Value
000	0
001	Multiplicand
010	Multiplicand
011	Multiplicand * 2
100	Multiplicand * -2
101	Multiplicand * -1
110	Multiplicand * -1
111	0

Wallace tree adder performs the addition in parallel, unlike the carry save addition where the sequential addition takes a longer time The idea of Wallace tree addition is depicted in the following figure

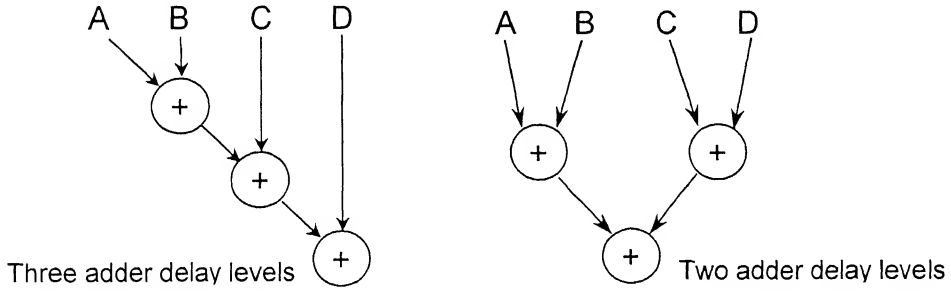


Figure 11 Parallel Adder Tree

Putting the suggestions by Booth and Wallace together, the partial products are generated with the Booth recoding scheme and the additions are performed with the Wallace tree adders The modules for both of the tasks are designed separately and instantiated in the final design as components

The schematics of the Booth-Wallace multiplier considered in the present work is shown below

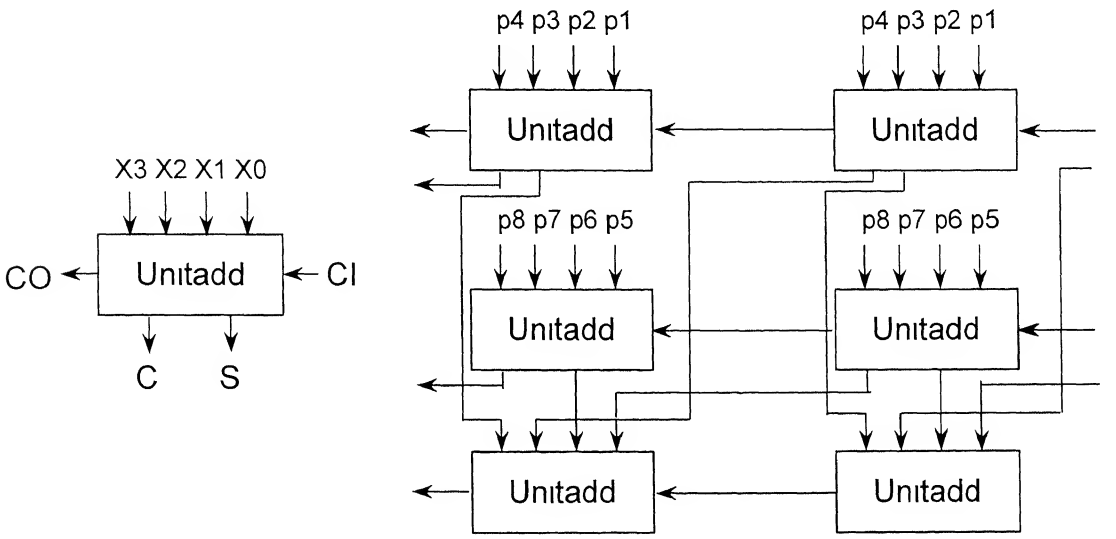


Figure 12 Adder Organization for Partial Products

The Booth-Wallace multiplier is used for high-speed multiplication at the cost of hardware overhead

Q15 Binary Division

One of the simplest methods of binary division is the digit-by-digit algorithm, which is similar to the normal decimal approach to decimal numbers The circuit used for multiplication can also be modified to perform division Partial products as in division will now be partial remainders, which will be compared against the shifter divisor

The pair of n-bit shift register holds the partial remainders To start with, these register hold the dividend The divisor stays in another n-bit register throughout the process In each step the contents of the register-pair is shifted to the left

and the vacant cells are used to hold the remainder When the division process terminates, both remainder and quotient are obtained from the registers

The schematics for the divider considered in the work is shown below

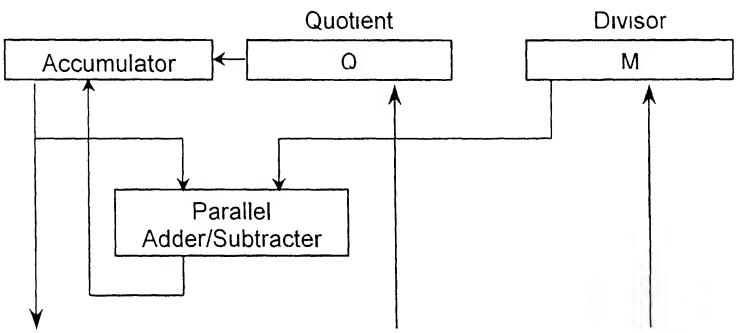


Figure 13 Sequential n-bit divider block

Binary division, unlike multiplication has other implications too The design considered for the work incorporates the check for a zero divisor condition and overflow, which occurs when divisor is very small compared to the dividend

Chapter 3

Methodology

This chapter introduces the approach of the present work towards the outlined objective It encompasses elaboration of various steps in the design process along with the respective methods and tools used

As system complexity increases, a high-level, top-down design approach becomes essential Therefore, the understanding of the top-down design process, and the effective use of the hardware description language VHDL become important The present work adheres to the top-down design approach

The behavioral description of the arithmetic unit, modeled using VHDL serves as the basis for the design This behavioral description is verified using the VHDL test bench The RTL modeling of various implementations all four arithmetic operations was done separately The choice of algorithm to be incorporate in the final design is made based on the results of comparative study of the FPGA implementations of these arithmetic operations The work comprises of the following steps

- Behavioral modeling & test bench verification
- RTL modeling & simulation
- Netlist extraction & place and route

Behavioral modeling

Behavioral approach to modeling hardware components is the key to the top-down design process It accurately models the input output behavior of the hardware, an arithmetic unit in the present work, but doesn't take into account the implementation of the functionality of the design The behavioral model of a component can be used anywhere in full design through instantiation and can be left as is until the structure of the component if finalized

In the present work, to start with, the arithmetic unit is partitioned into various functional units and behavioral model of each unit is prepared The following figure shows the functional decomposition of the arithmetic block

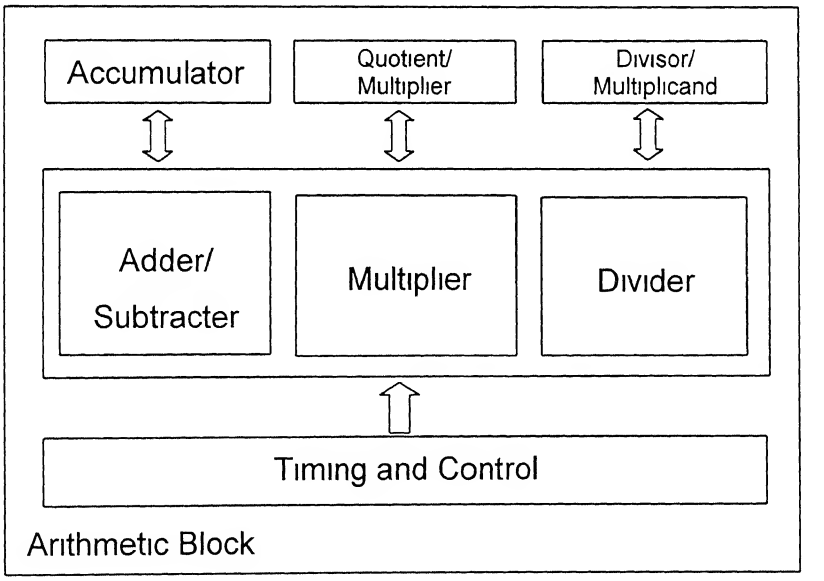


Figure 14 Functional Units in Arithmetic Block

The tool used for behavioral modeling is Active HDL 4 0, which complies by IEEE 1076 VHDL '93 standards and has IEEE 1164 extension to use multi-value logic that is required for accurate modeling of systems

Each functional module in the arithmetic unit was modeled separately and was instantiated as component at the time of integration The registers used in the design were designed at the gate level using the hierarchical arrangement of gates and flip-flops

The control unit of the arithmetic block makes use of **process** statement in VHDL to call the desired routine through **switch case** statement The following opcode list enumerates the functions

00	Addition
01	Subtraction
10	Multiplication
11	Division

Once behavioral modeling is over it has to be verified, using the test bench, for correctness of the design

Test bench verification

A test bench is an environment where a design is checked by applying signals and monitoring its responses In VHDL, test bench is just like another specification and is simulated to run the testing process

The verification of the behavioral model of the arithmetic unit was done in two steps first, each functional block is verified separately and then after integrating them as one entity, the arithmetic unit The test bench for the integrated

arithmetic unit was written to import test vector from a file on disk The test vector file was kept for the verifications at the later stages of designing

RTL modeling

Once behavioral description of the arithmetic is done and is verified, RTL modeling of its constituent functional block is performed Asynchronous designs for various fixed-point binary arithmetic operations in Q15 format were described at RTL level in the VHDL

RTL description of the functional blocks was again made using the top-down design approach Any function that happens to occur at a number of places in the design, like the carry look-ahead generation for a nibble in the addition/subtraction block, was modeled separately and instantiated in the design

The fixed-point binary arithmetic operations that were described at RTL level are enumerated as under

- Addition/Subtraction
- Multiplication
- Division

Out of the various implementations for the arithmetic operations, a selection has to be made as to which implementations suits the requirement This is done after performing a comparative study between the FPGA implementations followed by the place and route of the RTL descriptions The area and timing results were used to discriminate between the implementations

The RTL designs are simulated to verify their correctness Simulations are run in the same VHDL environments, the stimuli can be applied and the responses can be monitored over the simulation time

Simulation

Active HDL 4 1 has an integrated waveform editor, which is used to monitor the signal waveforms over the simulation time This facilitates debugging and provides a visual aid to observe the behavior of the design The following figure shows the waveform editor available with the VHDL tool

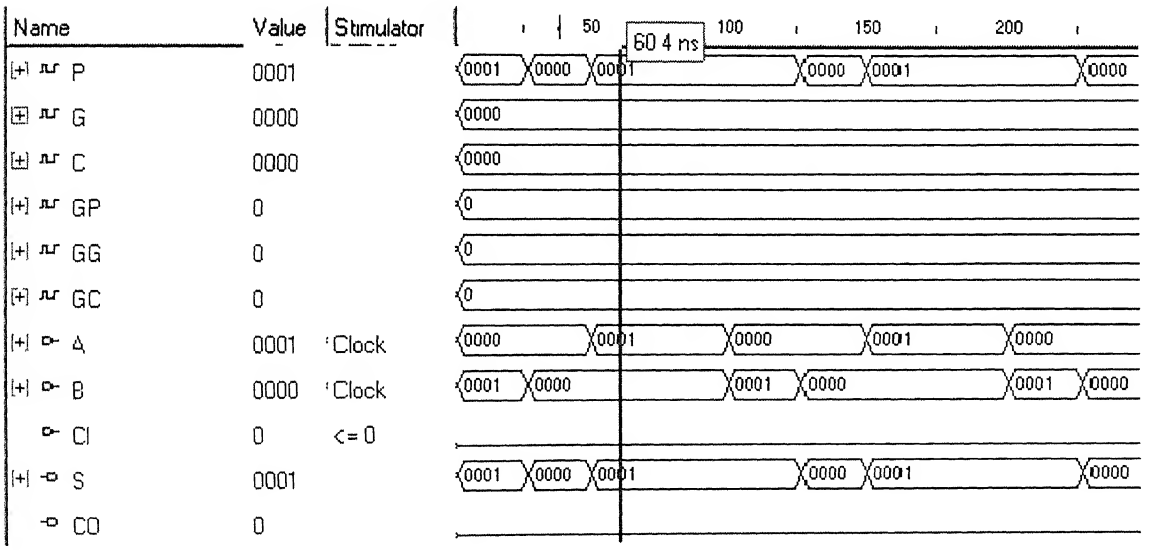


Figure 15 Active HDL Waveform Editor

Once the designs are simulated and verified for correctness, FPGA synthesis is performed

FPGA synthesis

In the present work Synopsys FPGA Compiler II version 2000 03 that is a complete logic-synthesis, optimization, and analysis CAD tool The place and route tool governs the choice of target architecture for design optimization In the present work, Xilinx XC 4000 family of FPGA was used as the target device
The following figure summarizes the FPGA Compiler II design overview

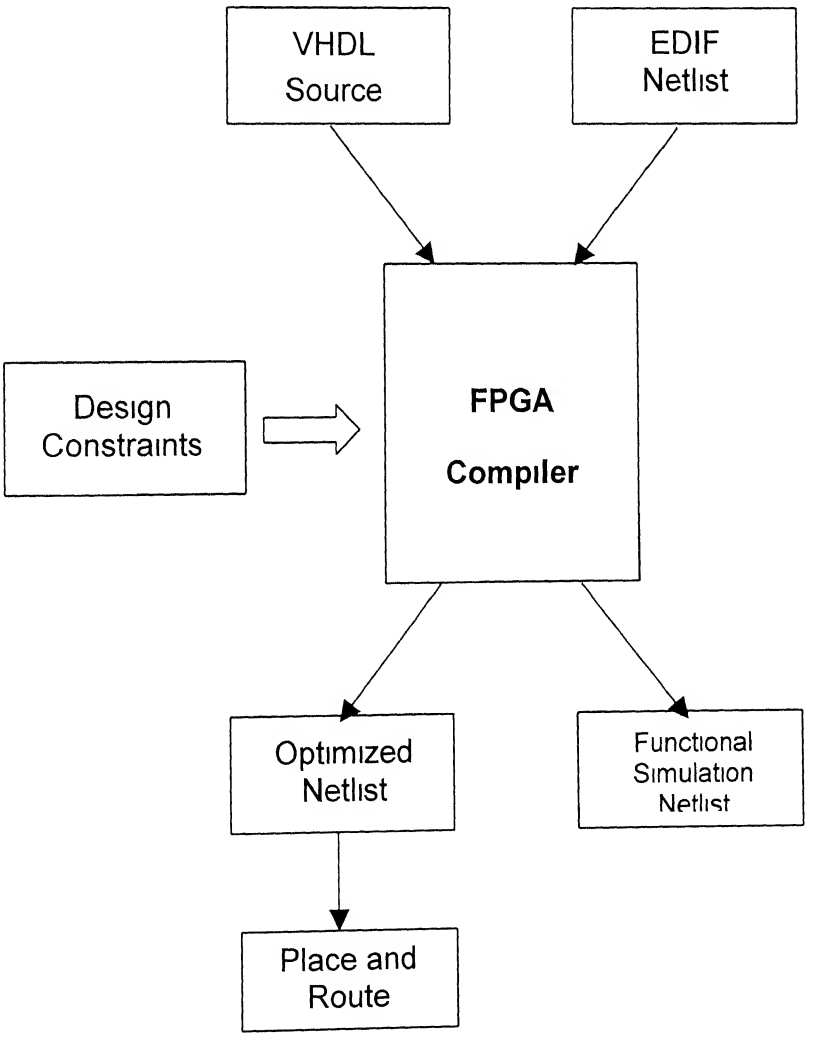


Figure 16 FPGA Compiler Design Flow

The RTL descriptions were imported to the compiler and Xilinx XC4005XL was selected as the target device After optimization, an estimate of area and timing results along with the schematics become available Detailed path and group delays are reported as the result and based on these results design was modified and optimized

More exact information about the area and timing can be obtained after place and route The netlist are exported from the FPGA compiled II for the use with place and route tool

Netlist Extraction

With the FPGA compiler II, various forms of netlist can be extracted A VHDL netlist can be utilized for functional verification, while an EDIF netlist can be mapped to a standard cell library In the present work, EDIF netlist were extracted for all the RTL level designs This EDIF netlist works as the basic input for the place and route tool

Place and Route

The CAD tools used for place and route is the Xilinx Foundation Series Student Version 1 5 The features of the place and route tools motivated the choice of target architecture to be Xilinx XC4013XL

It takes the standard EDIF netlist as input and produces a placed and routed version of the design Exact area and timing results area available after this step The Equivalent gate count, available after place and route serves as the basis for the area considerations of the module

Based upon the results, the implementations of various arithmetic operations are be discriminated and an optimal selection between them is be made

Chapter 4

Results and Discussion

In the present work, the following algorithms are considered for implementation

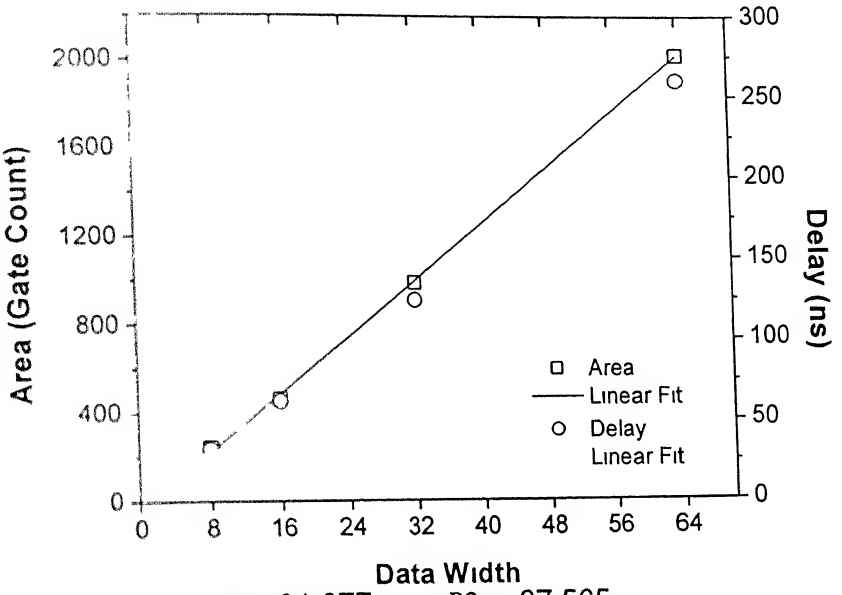
- Ripple Carry Addition
- Look Ahead Carry Addition
- Carry Skip Addition
- Shift and Add Multiplication
- Carry Save Multiplication
- Booth Wallace Multiplication
- Sequential Restoring Division
- Non-Restoring Division

Following are the results of the comparative study, between the different implementations of binary arithmetic operations The plots depict the dependence of area, in terms of gate count, and maximum delay over the data width In the plots, the dependence of area and delay over the data width are shown as the data points and the solid and dotted lines depict the fitted curve for area and delay, respectively The equations used for curve fitting

Linear Fit	$y = P1 * x + P2$
Logarithmic Fit	$y = P1 * \log_2(x) + P2$
Square Fit	$y = P1 * x^2 + P2$
Square Root Fit	$y = P1 * \sqrt{x} + P2$

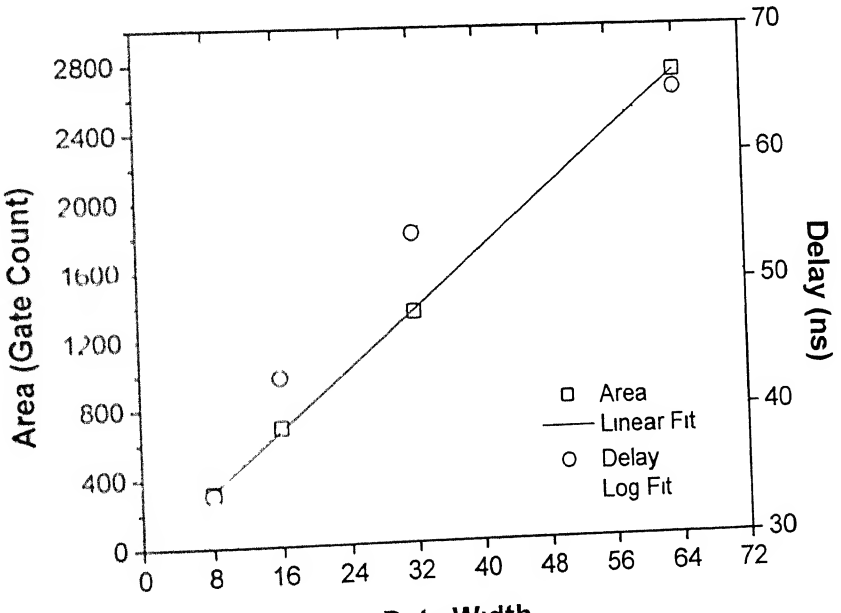
The values of fitting parameters $P1$ and $P2$ are given with each plot

Ripple Carry Adder

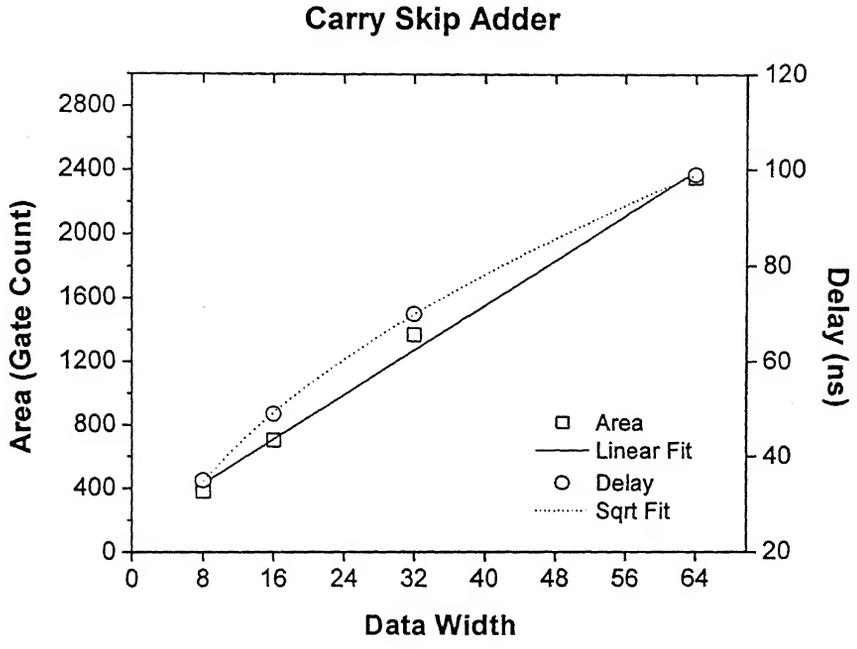


Area	$O(n)$	$P1 = 31\ 977$	$P2 = -27\ 565$
Delay	$O(n)$	$P1 = 4\ 084$	$P2 = -3\ 043$

Look Ahead Carry Adder

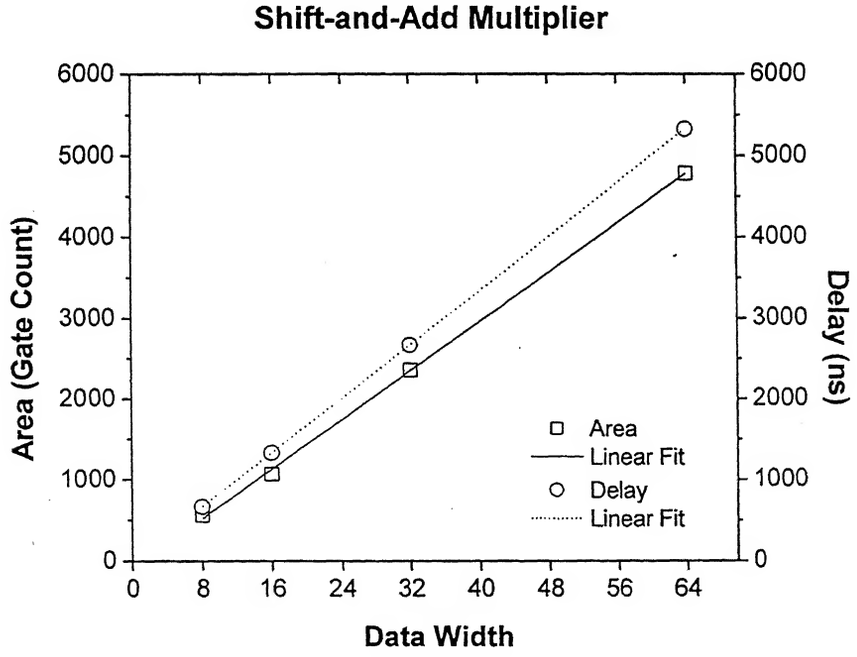


Area	$O(n)$	$P1 = 42\ 910$	$P2 = -19\ 823$
Delay	$O(\log_2 n)$	$P1 = 10\ 413$	$P2 = 2\ 201$



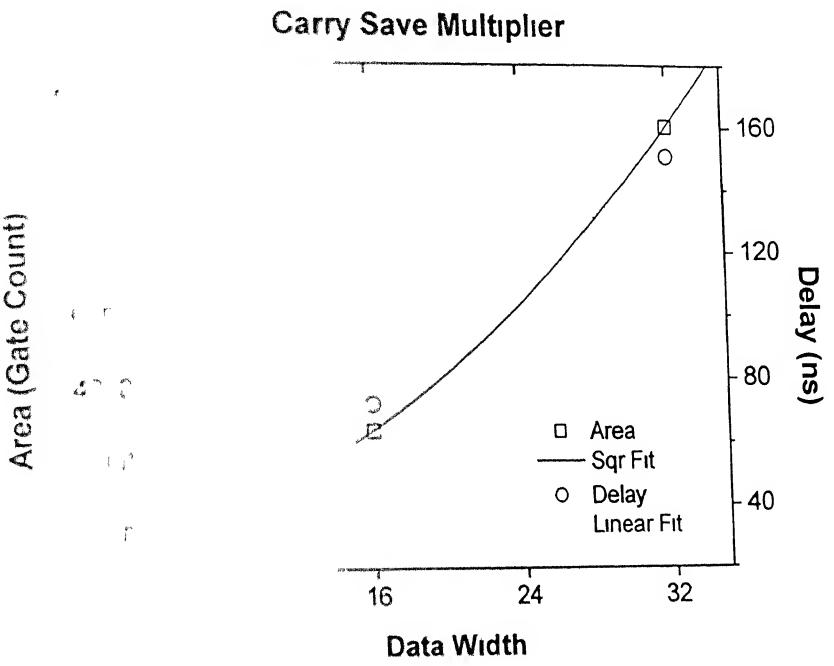
Area $\equiv O(n)$ $P1 = 35.051$ $P2 = 150.216$

Delay $\equiv O(n^{1/2})$ $P1 = 12.412$ $P2 = -0.317$

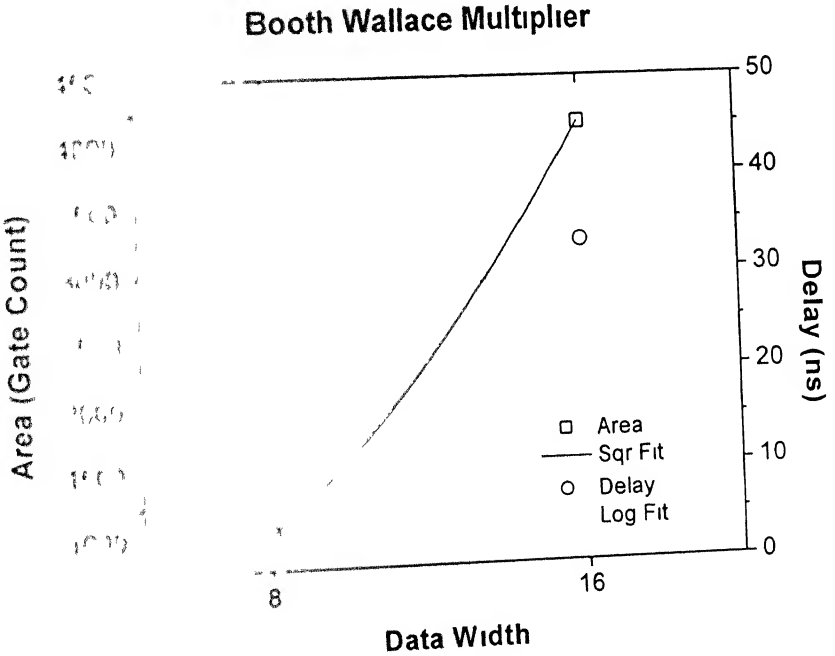


Area $\equiv O(n)$ $P1 = 76.075$ $P2 = -90.779$

Delay $\equiv O(n)$ $P1 = 83.234$ $P2 = 0.365$

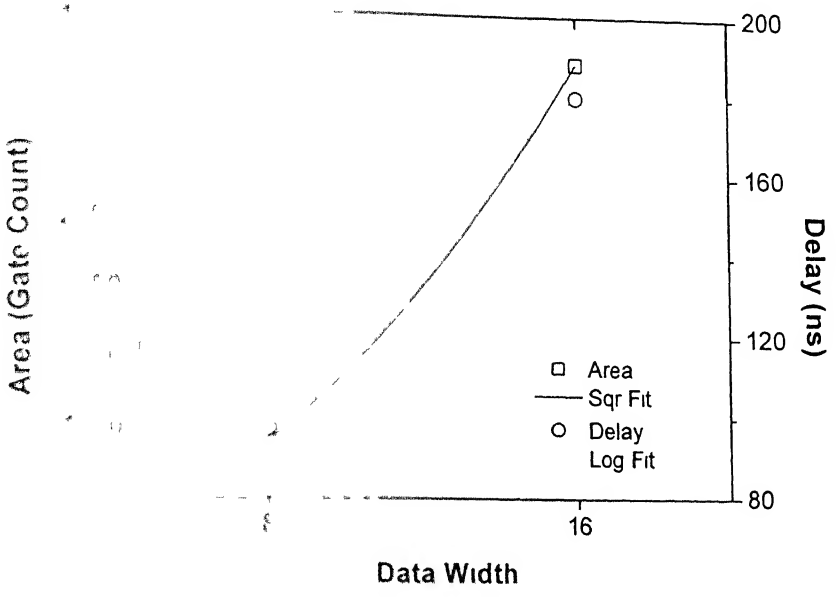


Area	$O(n^2)$	$P1$ = 11 017	$P2$ = 9 667
Delay	$O(n)$	$P1$ = 4 741	$P2$ = -1 500



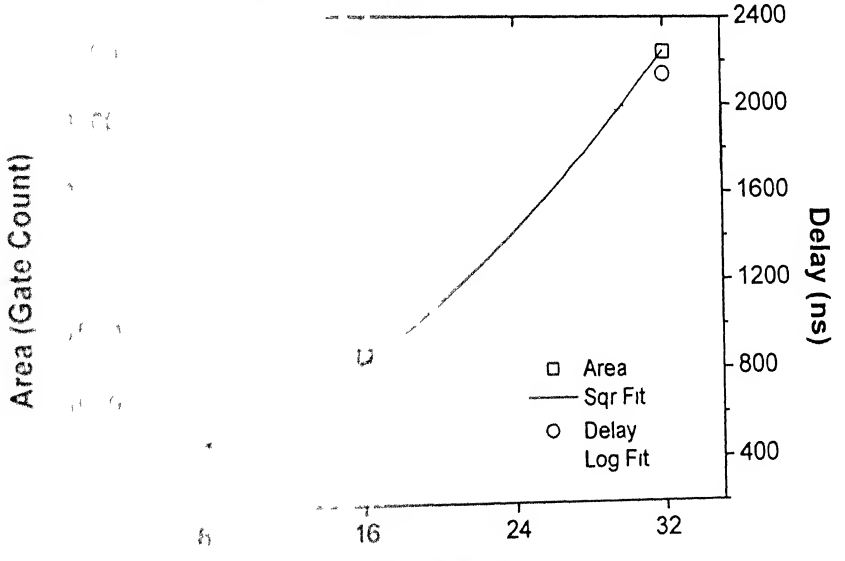
Area	$O(n^2)$	$P1$ = 16 156	$P1$ = 2 080
Delay	$O(\log\ n)$	$P1$ = 16 001	$P1$ = -31 108

Non-restoring Divider



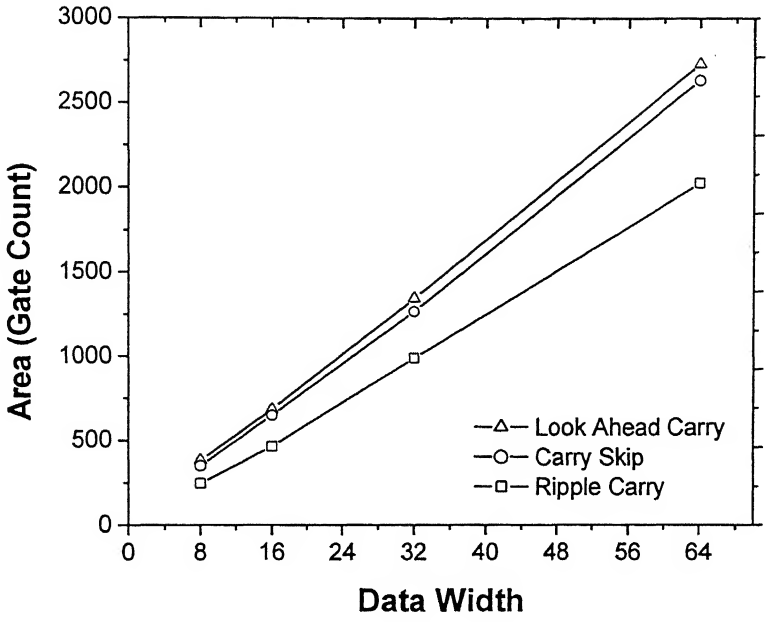
Area	$O(n^{\cdot})$	$P1 = 14\ 302$	$P2 = 3\ 248$
Delay	$O(\log^{\cdot} n)$	$P1 = 83\ 124$	$P2 = -152\ 014$

Sequential Restoring Divider



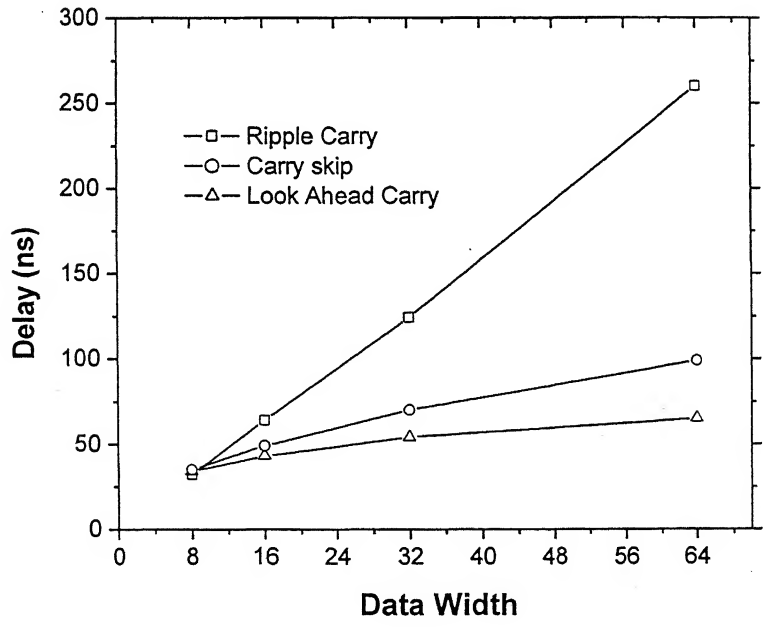
Area	$O(n)$	$P1 = 2\ 920$	$P2 = 512\ 829$
Delay	$O(\log n)$	$P1 = 841\ 499$	$P2 = -2130\ 330$

Adder Area Comparison



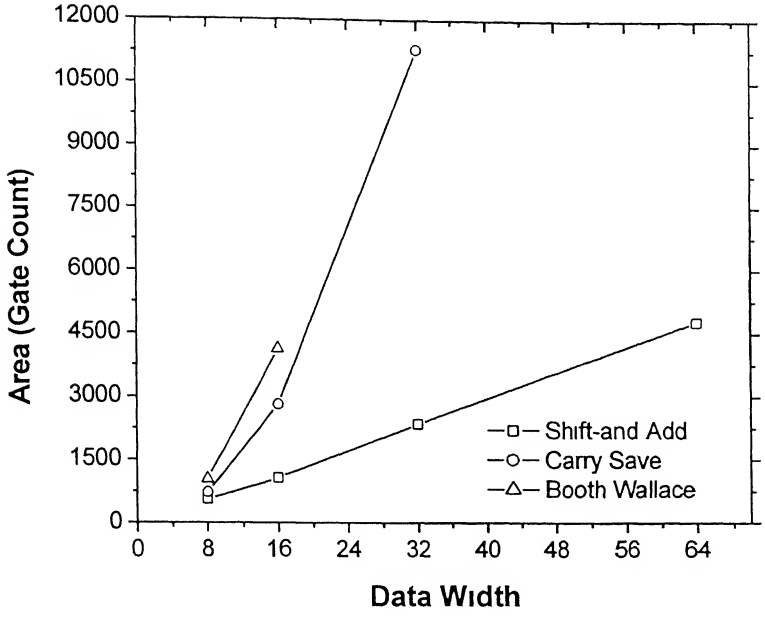
Maximum Area ≡ Look-Ahead carry Adder
Minimum Area ≡ Ripple Carry Adder

Adder Delay Comparison



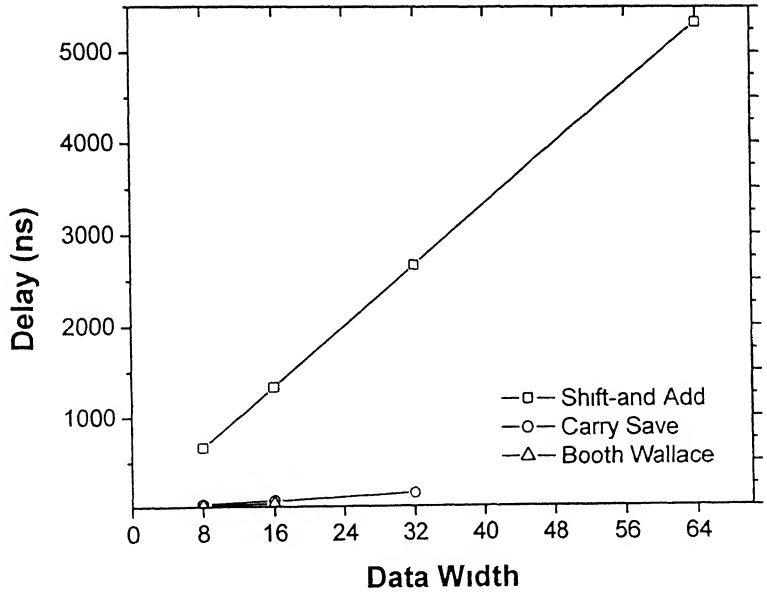
Maximum Delay ≡ Ripple Carry Adder
Minimum Delay ≡ Look Ahead Carry Adder

Multiplier Area Comparison



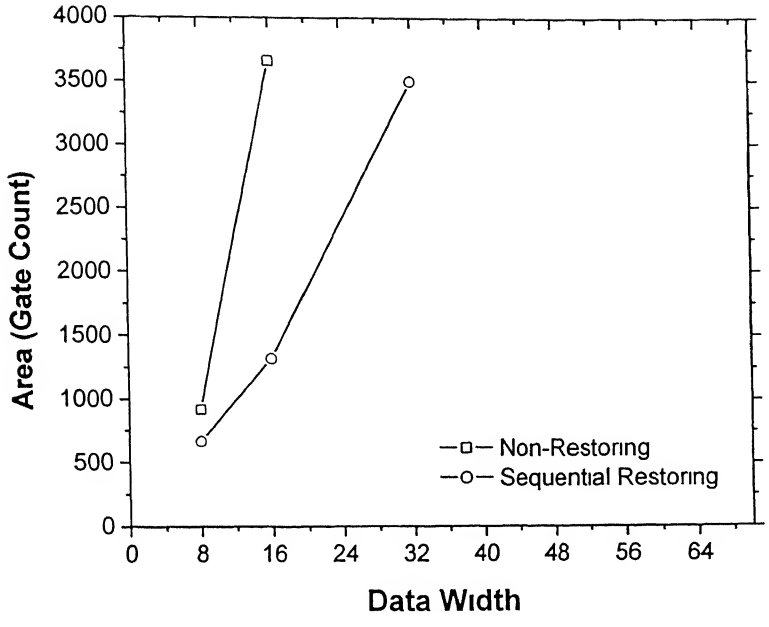
Maximum Area ≡ Booth Wallace Multiplier
Minimum Area ≡ Shift-and-Add Multiplier

Multiplier Delay comparison

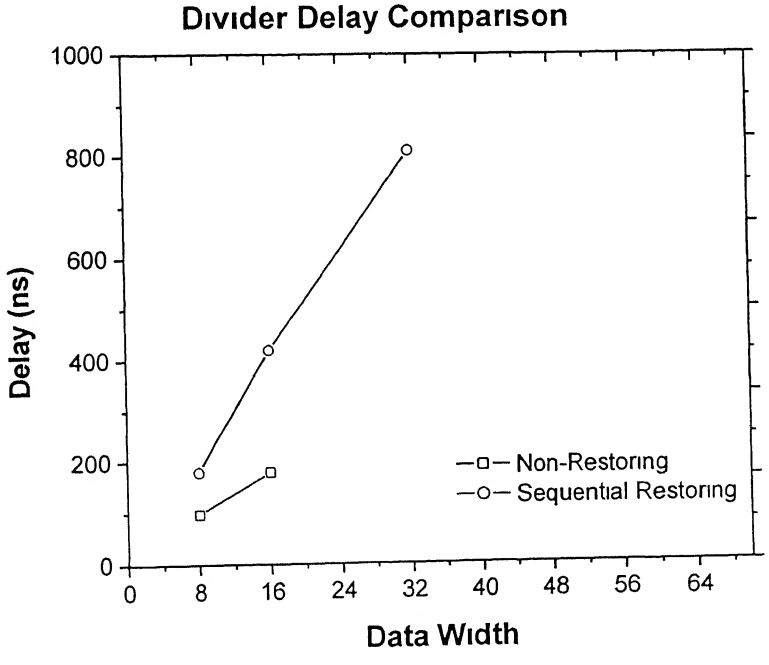


Maximum Delay ≡ Shift-and-Add Multiplier
Minimum Delay ≡ Booth Wallace Multiplier

Divider Area Comparison



Maximum Area ≡ Non-Restoring Divider
Minimum Area ≡ Sequential Restoring Divider



Maximum Delay ≡ Sequential Restoring Divider
Minimum Delay ≡ Non-Restoring Divider

Operation	Module Name	Max Delay (ns)	Max Freq (MHz)	Gate Count	Chip Usage
Addition/ Subtraction	Ripple Carry	62 379	16 031	466	10 %
	Look Ahead	43 167	23 165	684	14 %
	Carry Skip	49 281	20 291	704	14 %
Multiplication	Shift and Add	80 293x16	12 545	1069	22 %
	Carry Save	71 918	13 904	2816	57 %
	Booth Wallace	33 180	30 138	4138	82 %
Division	Sequential	69 426x16	14 403	1317	27 %
	Non Restoring	180 100	5 55	3662	72 %

Q15 Arithmetic Implementation over XC4005XL FPGA

Discussion

In the present work, the standard results for each implementation are taken from Computer Arithmetic [12], Integrated Systems Laboratory ETH Zurich All the implementations were found to be in accordance with the expected values The final selection of the arithmetic algorithms for the design of arithmetic block is made based on the results of respective implementation

The entire individual arithmetic blocks were synthesized over Xilinx FPGA XC4005XL and the area usage for optimal implementations for multiplier and divider block were found to be close to the maximum possible gate count Owing to this, the final integrated arithmetic block was synthesized over XC4013XL FPGA

For Addition/subtraction, the look-ahead carry adder provided a speed-up over the ripple carry adder, at the cost of a medium increase in hardware. Carry skip adder, due to inherent logic redundancy and the subsequent difficulties in design optimization was not considered in the final arithmetic block.

For Multiplication, Booth Wallace multiplier was considered in the final design for its high speed, though it was costly in terms of gate count compared to carry save and shift-and-add multipliers.

For division, though the area requirements of non-restoring divider are high, it is considered for the final implementation of arithmetic block. Sequential restoring divider was found to be much slower compared to non-restoring divider circuit.

Chapter 5

Conclusion and Future Scope

Conclusion

The present work involved the development of the core design of fixed-point arithmetic block The design supported the basic arithmetic operations of addition, subtraction, multiplication, and division of fixed-point binary numbers in Q15 format Further, the design was modeled so that it is extendable to Q31 format arithmetic operations The generality and portability is maintained by avoiding any vendor-specific optimized block in the design Based upon the results of the comparative study of the different implementations of binary arithmetic operations, the optimal algorithms were selected and integrated to form a complete arithmetic block

Future Scope

The following points can be enumerated towards the future scope for the present work

- Bit-sliced architectures for addition/subtraction may be explored for better implementation results

Vendor-specific optimized blocks could be used to obtain better implementation results

Device could be designed and optimized for a preset data width to facilitate the optimization of the code, which for generalized data width is not as lucid

With the availability of CAD tools, the design can be synthesized with different target architectures to explore the possibilities of better implementations

Other useful arithmetic operations like square and square root could be incorporated in the design

Bibliography

1 Ashenden Peter J , The VHDL Cookbook, Computer Science Dept University of Adelaide, 1990

2 Brown et al , VLSI Circuits and Systems in Silicon, McGraw-Hill, 1991

3 Gaonkar R S , Microprocessor Architecture, Programming and Applications, Penram International, 1997

4 Hayes J P , Computer Organization and Architecture McGraw-Hill, 1988

5 Giacomo Joseph Di, VLSI Handbook, McGraw-Hill, 1989

6 Chang K C , Digital Systems Design with VHDL and Synthesis, IEEE Computer Society, 1999

7 Weste Neil H E and Eshraghian Kamran, Principles of CMOS VLSI Design, Addison-Wesley, 1998

8 Murgai Rajeev, Brayton Robert K , Alberto S , Logic Synthesis For Field-Programmable Gate Arrays, Kluwer Academic, 1995

9 Synopsys FPGA Compiler II, User Manual

10 Wolf Wayne, Modern VLSI Design, PTR Prentice Hall, 1994

11 Xilinx Foundation Series Student Version, User Manual

12 Reto Ziemerrmann, Computer Arithmetic Principles, Architectures, and VLSI Design, Integrated Systems Laboratory, Swiss Federal Institute of Technology 1991